Indexing of sequential patterns for efficient analysis of data mining results

Witold Andrzejewski, Tadeusz Morzy, and Mikołaj Morzy

Institute of Computing Science
Poznań University of Technology
Berdychowo, 60-965 Poznań, Poland
{wandrzejewski,tmorzy,mmorzy}@cs.put.poznan.pl

Abstract. Mining sequential patterns allows the analysis of large databases of sales data that are not susceptible for investigation "by hand". Unfortunately, data mining algorithms produce a large number of results, that often exceed the size of the original database. Analysis of data mining results is usually performed by the end user of the data mining application manually. During this process a large number of subsequence queries are executed. Such queries are not well supported by traditional database management systems. In this paper we present a novel indexing technique for sequences of sets such as sequential patterns or sales history. Experimental evaluation of the index proves the feasibility and benefit of the index in exact and similar matching of subsequences.

1 Introduction

Popularity of database systems, data warehouses, and other data repositories has become the cause of amassing large volumes of data. It is estimated that, since the year 2000, the size of the stored data has increased two times. Each year, 5 million terabytes of new data are stored. Analysis and effective access to such massive volumes of data have become a crucial problem today.

Large volumes of data cannot be effectively analysed "by hand". Instead, numerous techniques for *knowledge discovery in databases* (also known as *data mining*) have been developed. Data mining is a process of discovering new, useful, correct and understandable patterns in very large volumes of data. Discovered patterns facilitate developing better plans and business strategies that increase company revenues, help in analysis of observed phenomena, and allow to formulate new theories.

Data mining technology allows to analyse many different types of data, such as: cash register data, stock prices or web server logs. Particularly interesting is the analysis of sales data, also known as *market basket analysis*. Through market basket analysis one may obtain association rules or sequential patterns. Association rules show which products are often purchased together and are represented as implications ($X \rightarrow Y$). Discovery of such a rule indicates, that if the customer buys products from the set X, then there is a high probability that she will also buy products from the set Y. Sequential patterns show which products are

purchased by a single customer over a period of time and are represented as a sequence of sets ($\langle X_1, X_2, X_3, \ldots, X_n \rangle$). Discovery of such a pattern indicates that if the customer buys products from the set X_1, then, after some time, she might buy products from the set X_2, then from X_3 and so on. Unfortunately, very often the number of discovered association rules, or sequential patterns, is larger than the size of the analysed database. Results of data mining may be therefore stored in a separate database for further analysis *off-line* [9]. Such an analysis is performed through retrieval and investigation of interesting rules in detail, as well as examination of customer transactions that either support, or violate discovered association rules or sequential patterns. During this process, a large number of subsequence or subset queries are executed. Such queries are not well supported by traditional database management systems.

Retrieval of sets has been widely investigated in the literature and many indexing schemes have been developed, such as: signature files [10], inverted files [4], RD-Trees [8], S-Trees [6] or hierarchical bitmap index [3]. On the other hand, several indexing schemes for sequences of atomic values have been proposed so far, such as: ISO-Depth index [24], SEQ-Join index [15] or SEQ family of indexes [18]. Alas, no proposals were given for indexing of sequences of sets.

In this paper we propose a new indexing scheme capable of efficient retrieval of sequences of sets based on non-contiguous subsequence containment and similarity. We present the physical structure of the index and we develop algorithms for query processing based on subsequence matching and subsequence similarity.

The rest of the paper is organized as follows. In Section 2 we introduce basic definitions used throughout the paper. Section 3 contains an overview of the related work. We present our index in Section 4. Experimental evaluation of the index is presented in Section 5. Finally, the paper concludes in Section 6 with a summary and a future work agenda.

2 Basic Definitions

An *element of a sequence* is a pair $S_i = (v(S_i), ts(S_i))$, where $v(S_i)$ denotes the *value* of the element, and $ts(S_i)$ denotes the *timestamp* of occurrence of the element S_i. A *sequence* S is an ordered set of elements S_i arranged according to their timestamps $ts(S_i)$. We define the *duration* of a sequence S as $ts(S_n) - ts(S_1)$ where S_n is the last element of the sequence S. A *subsequence* S' of the sequence S is a sequence created from the sequence S by removing arbitrary elements. A sequence $S' = \langle (v(S'_1), ts(S'_1)), \ldots, (v(S'_k), ts(S'_k)) \rangle$ is called a *continuous subsequence* of a sequence $S = \langle (v(S_1), ts(S_1)), \ldots, (v(S_n), ts(S_n)) \rangle$ (denoted $S' \subset S$) if

$$\exists w : \forall i = 1, \ldots, k \;\; v(S_{i+w}) = v(S'_i) \wedge ts(S_{i+w}) = ts(S'_i)$$

A sequence Q such that the first element of Q has the timestamp $ts(Q_1) = 0$ is called a *query sequence*. Each query sequence Q has a *tolerance sequence* T associated with it. The tolerance sequence T has the same cardinality as the query sequence Q. The elements of the tolerance sequence T are numbers, and their timestamps are consecutive integers. The elements of the tolerance sequence

T form tolerance ranges for corresponding elements of the query sequence Q of the form $(ts(Q_i) - v(T_i), ts(Q_i) + v(T_i))$. In addition, tolerance ranges must not disturb the order of elements, i.e., $ts(Q_i) + v(T_i) < ts(Q_{i+1}) - v(T_{i+1})$.

An *allocation* $A(Q, S')$ is a mapping of every query sequence element to an element of S' such that $\forall i = 1, \ldots, |Q|\ \ ts(S'_i) - ts(S'_1) - ts(Q_i) \in \langle -v(T_i), +v(T_i) \rangle$.

Given a query sequence Q, the *subsequence query* retrieves all sequences S having a subsequence S', such that the following condition is fulfilled

$$l = n \wedge \forall i = 1, \ldots, n\ \ v(Q_i) \subset v(S'_i) \wedge ts(S'_i) - ts(S'_1) - ts(Q_i) \in \langle -v(T_i), +v(T_i) \rangle$$

Let ϵ denote the threshold value of minimum similarity between two sequences. Given an allocation $A(Q, S')$ of the query sequence Q to the sequence S'. The *similarity query* retrieves all sequences S such that $\exists S' \subset S : sim(Q, S') > \epsilon$, where $sim(x, y)$ is any measure of similarity between two sequences.

Given a sequence of sets S, let $sig(S_i)$ denote a binary signature of the set $v(S_i)$ and let $sig(S) = \{sig(S_i) : i = 1, \ldots, n\}$ denote the set of signatures of all sets in the sequence S.

3 Related Work

3.1 Indexing sequences

Most research on indexing of sequence data focused on three distinct areas: indexing of time series, indexing of strings of symbols, and indexing of text. Most indexes proposed for time series support searching for similar or exact subsequences by exploiting the fact, that the elements of the indexed sequences are numbers. This is reflected both in index structure and in similarity metrics. Most popular similarity metrics include Minkowski distance [11, 26], compression-based metrics [12], and dynamic time warping metrics [23]. Often, a technique for reduction of the dimensionality of the problem is employed, such as discrete Fourier transform [1, 7]. String indexes usually support searching for subsequences based on identity or similarity to a given query sequence. Most common distance measure for similarity queries is the Levenshtein distance [13], and index structures are built on suffix tree [17, 21, 22, 25] or suffix array [16].

Indexing of sequences of symbols differs significantly from indexing of strings. The main difference is the fact, that symbols in a sequence of symbols are assigned a timestamp that must be taken into consideration when processing a query. Most proposals for indexing of sequences of symbols transform the original problem into the well-researched problem of indexing of sets [18]. The transformation of a sequence into a set first maps all sequence elements into set elements, and then adds additional elements representing the precedence relation between the elements of the original sequence. The main drawback of this technique is the fact, that it ignores the timestamps associated with sequence elements. This leads to an additional verification phase, where sequences returned from the index are verified against the query sequence to prune false hits.

ISO-Depth index [24] is an indexing structure that efficiently supports searching of sequences based on subsequence containment and similarity. ISO-Depth index stores all continuous subsequences of given length in a trie structure. Additionally, trie nodes are numbered in a way permitting to quickly determine the nature of the relationship between the nodes. The order of the nodes in the trie corresponds to the order of symbols represented by those nodes in sequences pointed at in the trie leaves. Diversification of symbols in the trie (symbols differ depending on the distance from preceding symbols in a sequence) allows to answer queries containing timestamp constraints. After creating the trie structure, ISO-Depth lists and position lists are read off the trie to form the ISO-Depth index.

An interesting proposal of SEQ-join index was presented in [15]. This index uses a set of relational tables and a set of $B+$-tree indexes. Each table corresponds to a single symbol appearing in the indexed sequences and contains ordered timestamps of the symbol together with a pointer to an appropriate sequence. Preparing a subsequence query consists in creating a directed graph with nodes representing query sequence elements and edges representing order constraints between sequence elements. Answering a subsequence query consists in performing a join between symbol tables using $B+$-tree index joins. Detailed description of subsequence query algorithms using SEQ-join is presented in [15].

3.2 Sequential patterns

Problem of mining sequential patterns was formulated by Agrawal and Srikant in [2]. Many algorithms for sequential pattern mining have been developed, such as: GSP [20], SPADE [27], SPAM [5], PrefixSpan [19] or MEMISP [14]. Sequential patterns are sequences that are frequently contained in sequences stored in the database. More formally, we say that a sequence S is *contained* in a sequence P if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $v(S_1) \subseteq v(P_{i_1}), v(S_2) \subseteq v(P_{i_2}), \ldots, v(S_n) \subseteq v(P_{i_n})$ where S_n is the last element of the sequence S. Note that even if the sequence S is contained in the sequence P, it does not mean that it is the subsequence of P, because the timestamps may not match. The *support* of the sequence S is the fraction of all the sequences in a database that contain the sequence S. We say that the sequence is *frequent* if it has support greater or equal to a user-specified threshold value *minsup*. The sequence is *maximal*, in a set of sequences, if it is not contained in other sequences in the set. Frequent sequences are called *sequential patterns*. Sequential patterns may be generalized to incorporate hierarchies of items and time constraints [20].

4 Generalized ISO-Depth Index

In this paper we extend the basic ISO-Depth index to support efficient indexing of sequences of sets. The new structure allows to search for similar subsequences and uses a similarity measure that is based on user-defined similarity measure for sets. We make no further assumptions on the similarity measure used to

compare sets that are elements of sequences, but we require the measure to (i) increase with the increase of the size of intersection of sets, and (ii) decrease with the increase of the Hamming distance between the sets.

To the best of authors' knowledge, there are no similarity measures for sequences of sets. Therefore we introduce two new measures that can be used when formulating similarity queries on sequences of sets. Given a query sequence Q and a subsequence S' of a sequence S, such that a valid allocation $A(Q, S')$ of Q to S' exists. *Liminal similarity* is defined as the minimum similarity between any pair of sets in the allocation. Formally,

$$sim_L(Q, S') = \min_{i=1,\ldots,|Q|} \{setsim(Q_i, S'_i) : (Q_i, S'_i) \in A(Q, S')\}$$

where $setsim(Q_i, S'_i)$ is the value of user-defined similarity measure for sets that fulfills the above mentioned requirements. *Average similarity* is the average similarity between all pairs of sets in the allocation $A(Q, S')$. This similarity is given by the formula below.

$$sim_A(Q, S') = \frac{1}{|Q|} \sum_{(Q_i, S'_i) \in A(Q, S')} setsim(Q_i, S'_i)$$

It is easy to notice that for any pair of sequences (Q, S') the value of the average similarity is always greater or equal to the value of the liminal similarity between the sequences.

Below we present the algorithm for constructing the Generalized ISO-Depth index. Given a database D consisting of n sequences S^k and the width of a moving window ξ.

1. For every sequence of sets $S^k \in D$ perform the following actions
   (a) Sequence S^k is transformed into a sequence of binary signatures B^k, such that $|\mathcal{S}^k| = |\mathcal{B}^k| \wedge \forall \mathcal{S}^k_i : \mathcal{B}^k_i = (sig(\mathcal{S}^k_i), ts(\mathcal{S}^k_i))$. Timestamp values should be discretized prior to building binary signatures. Query sequences should be transformed analogously. If the elements of indexed sequence do not have an explicit timestamp, numbers of elements may be used instead.
   (b) A moving window is used to read all continuous subsequences of B^k of the duration lesser or equal to ξ. For each such subsequence B'^k, the sequence identifier k is stored along with the position, where B'^k starts within B^k.
   (c) Subsequences B'^k are transformed into symbol sequences of the form x_i, where $x \in sig(S^k) \wedge i \in N \cup \{0\}$ using the function

$$f(B'^k) = \langle x_1, \ldots, x_n \rangle \text{ where: } x_i = \begin{cases} v(B'^k_i)_0 & \text{if } i = 1, \\ v(B'^k_i)_{ts(B'^k_i) - ts(B'^k_{i-1})} & \text{if } i > 1. \end{cases}$$

   (d) Symbol sequences created in the previous step are then inserted into a modified trie structure. We modify the original trie structure in the

following way: instead of defining an additional terminator symbol we add subsequence identifier to a trie node in which a given subsequence terminates. In general, there can be several subsequences terminating in a given node. Therefore, each node of the trie contains a list of subsequence identifiers.

2. The trie is traversed and all nodes are numbered using the depth-first search order. Additionally, each node is marked with the highest number of the node contained in a sub-trie starting at a given node. Those two numbers determine the range of node numbers contained in a given sub-trie. The distance of a given node from the beginning of the subsequence is simply the sum of indexes of symbols on the path to a given node.
3. The trie is used to extract ISO-Depth lists of elements of the form $(s, (a, b))$, where s is a signature of a set and the range (a, b) is the range of node numbers stored in the node pointed at by the edge representing the signature s. Each ISO-Depth list orders elements according to the value of a, and for all nodes stored in the list the distance of the node from the beginning of the subsequence is the same.
4. After creating ISO-Depth lists the trie is used to generate position lists. Each position list stores information corresponding to sequences that terminate in a given node. A position list is generated for each node where a sequence terminates.
5. ISO-Depth lists and position lists together form the Generalized ISO-Depth index. The trie structure is not used anymore and can be safely discarded.

Algorithms for processing of sequence-oriented queries using the Generalized ISO-Depth index use the following lemma.

Lemma 1. *Ranges of node numbers stored on a ISO-Depth list for a given distance from the beginning of the sequence are disjoint. Given ISO-Depth lists for distances $d_k < d_l$ from the beginning of the sequence. Let the entries on the lists be of the form $(s^k, (a^k, b^k))$ and $(s^l, (a^l, b^l))$, respectively. If $a^k < a^l \leq b^l \leq b^k$, then the database contains a sequence, such that a subsequence exists that contains sets with signatures s^k, s^l, respectively. Moreover, if the timestamp of the first element of this subsequence is subtracted from other timestamps of the subsequence elements, then the timestamps of those sets are d_k, d_l.*

The algorithm for processing of subsequence queries is given below. Let us assume that the query sequence is given as $Q = \langle (v(Q_1), 0), \ldots, (v(Q_n), ts(Q_n)) \rangle$.

1. For each timestamp $ts(Q_i)$ retrieve the ISO-Depth list for the distance equal to the timestamp.
2. Search the lists recursively. For each ISO-Depth list entry $(s^1, (a^1, b^1))$,check if the signature $sig(Q_1)$ is contained in s^1. If true, search the ISO-Depth list corresponding to the next element of the search sequence looking for an entry $(s^2, (a^2, b^2))$, such that $a^1 < a^2 \leq b^1$ and find signatures s^2 containing $sig(Q_2)$. For each such s^2 search the list corresponding to the next element of the query sequence retrieving only the entries contained in (a^2, b^2).

3. Continue this procedure until the last element of the query sequence is reached. Signatures retrieved during each recursive call, along with the timestamps corresponding to the subsequent ISO-Depth lists, form the searched subsequence.
4. If a signature s^n is found such that s^n contains $sig(Q_n)$, use position lists to find all pointers to subsequences stored in the nodes with numbers in the range (a^n, b^n). Store those pointers for the sake of future verification. Return to the recursive traversal of ISO-Depth lists.
5. Read the subsequences accessed via stored pointers to verify the actual subsequence containment (this is required due to ambiguity introduced by binary signature generation procedure).

Algorithms for subsequence similarity matching are similar to the algorithm presented above. We design two algorithms, one capable of using tolerance sequences when searching for a similar subsequence, and one used for strict similarity subsequence searches. Both algorithms use the upper bound of approximation of similarity between compared sequences. This approximation is based on the upper bound of the intersection and the lower bound of Hamming distance between sets that are elements of the compared sequences. Using this approximation allows significant pruning of sequences. The upper bound approximation is used during step (2) of the algorithm, instead of checking for the containment of $sig(Q_i)$ in s^i. For queries allowing tolerance sequences, the algorithm needs to retrieve in step (1) not only ISO-Depth lists for the distance equal to the timestamp $ts(Q_i)$, but all ISO-Depth lists for distances from the range $(ts(Q_i) - v(T_i), ts(Q_i) + v(T_i))$ and merge these lists into a single list.

5 Experimental Results

The efficiency of the index is experimentally evaluated and the results of the conducted experiments are presented below. For each experiment 40 different sequence databases were generated. Elements of sets contained in sequences were generated using homogeneous and Zipf distributions. Table 1 summarizes the parameters used in experiments.

After building indexes the sets of query sequences were generated. For each database 7 different sets of 10 query sequences were prepared. Each set consisted of subsequence queries and similarity queries (with and without tolerance) for similarity thresholds of 70%, 80%, and 90%.

Experiment 1 measures the efficiency of the index with respect to increasing the size of the database. Figure 1 presents the performance of the Generalized ISO-Depth index (using 8 bit and 16 bit signatures) for subsequence queries (*Subseq*), exact similarity queries (*Sim*), and similarity queries with tolerance (*Tol*). Figure 2 presents the results for the same queries without the index. It can be easily seen that the index is two to four orders of magnitude faster than the naive approach. Query processing time grows linearly with the number of sequences stored in the database. Indexes using 8 bit signatures are faster for all

Table 1. Synthetic data parameters

parameter	**Exp.1**	**Exp.2**	**Exp.3**
size of the domain	150 000	150 000	150 000
minimal distance between sets	1	1	1
maximal distance between sets	100	100	100
minimal set size	1	1	5–100
maximal set size	30	30	15–110
minimal number of sets in sequence	2	5–100	2
maximal number of sets in sequence	20	15–110	2
number of sequences	10 000–100 000	10 000	10 000
signature length	8b,16b	8b,16b	8b,16b
page/node size	4096B	4096B	4096B
window width (ξ)	250	250	250

classes of queries. We attribute this to the fact that shorter signatures induce smaller trie structure, less nodes in the trie, and shorter ISO-Depth lists. Of course, shorter signatures produce more ambiguity and more false hits have to be verified. Nevertheless, our experiments show that the benefit of using shorter signatures surpasses the cost of additional false hit verification.

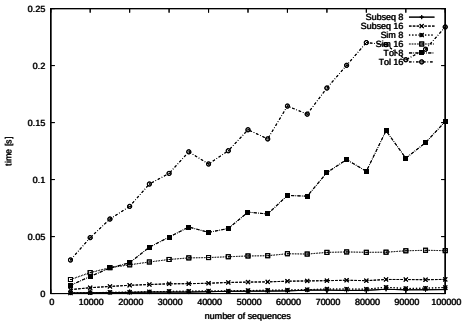

Fig. 1. Number of sequences

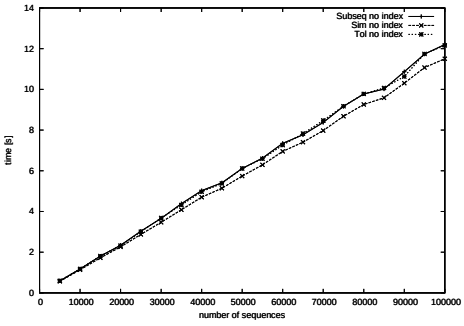

Fig. 2. Number of sequences (no index)

Experiment 2 studies the impact of the average number of sets in indexed sequences on the performance of the Generalized ISO-Depth index. We vary the average number of sets from 10 to 105. Figure 3 shows the performance of our index for three classes of queries. The results for the same queries without the index are depicted in Figure 4. Both figures exhibit the results similar to the results obtained in Experiment 1. This similarity can be easily explained. The number of subsequences inserted into the trie depends both on the number of sequences in the database, and the number of sets in indexed sequences. Conclusions of the Experiment 1 apply equally to the results of Experiment 2.

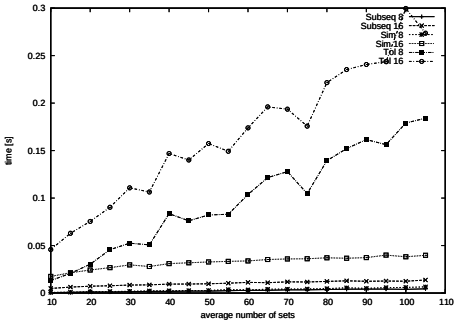

Fig. 3. Average number of sets

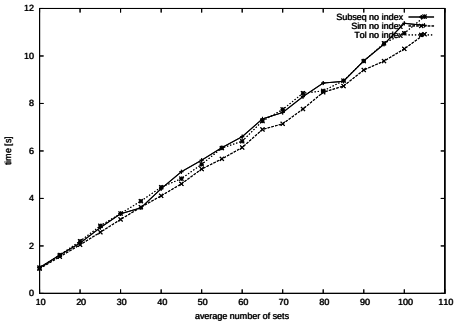

Fig. 4. Average number of sets (no index)

Experiment 3 measures the impact of the average size of sets being elements of the indexed sequences on the performance of the Generalized ISO-Depth index. We vary the average size of sets from 10 to 105. Figure 5 presents the results of three classes of queries when using the index, while Figure 6 shows the results of the same queries when not using an index. The shapes of curves presented in both figures can be easily explained. As the average size of a set grows, the probability that all positions of the signature corresponding to a given set would be set to '1' also increases. In other words, the increase of the average set size causes the saturation of signatures. Therefore, the diversity of signatures diminishes, and the set of all signatures stored in the trie becomes more compact. As the result, the number of nodes in the trie decreases and ISO-Depth lists become shorter. This in turn results in shorter processing times, although increases the number of false hits that need to be pruned. As we have already mentioned, our experiments suggest that this additional verification phase still pays off because of the shortened access time. After reaching a certain threshold, the signatures are fully saturated with bits set to '1' and the processing time stabilizes.

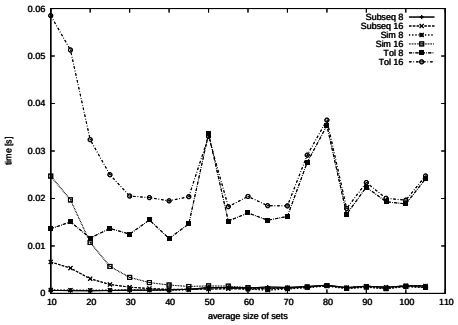

Fig. 5. Average size of sets

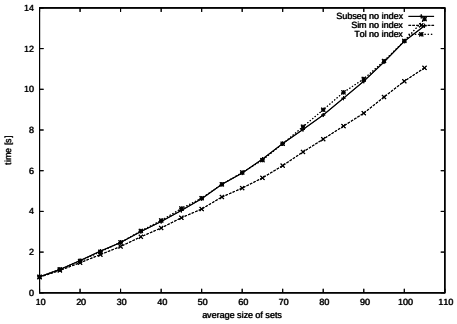

Fig. 6. Average size of sets (no index)

6 Conclusions

To the best of authors' knowledge, Generalized ISO-Depth index presented in this paper is the only index structure for sequences of sets proposed so far. Our index supports different classes of sequence-oriented queries, such as subsequence queries and similarity queries. The experiments show that the ratio of speed-up for those queries is two to four orders of magnitude when compared to brute-force approach. Possible applications of Generalized ISO-Depth index include, but are not limited to, indexing of sequential patterns, indexing of customer purchase data, indexing of multimedia databases, or analytical processing systems.

Some optimizations are still possible. Notice that ISO-Depth list for the distance 0 is read during every query execution. Therefore, it should be kept in memory to reduce the number of disk reads.

Our index has one problem when it comes to indexing sequences that do not have elements with explicitly specified timestamps. As we have suggested in Section 4, number of elements in a sequence may be used instead of missing timestamps. The problem is that the index supports only finding sequences that have a given subsequence, but not finding sequences that contain the given non-timestamped sequence. For example, let the database contain a sequence $S^k = \langle(\{1,2\},1),(\{3,4\},2),(\{5,6\},3)\rangle$. User wants to find all sequences that contain the sequence $Q = \langle\{1\},\{5\}\rangle$. According to the definition of containment from the Section 3.2 the sequence S^k contains the sequence Q. However, when, instead of timestamps, numbers of elements are used, the sequence $Q = \langle(\{1\},0),(\{5\},1)\rangle$ (query sequence needs to have the first timestamp equal to 0). When a query is executed, the sequence from the database will not be found, because the sequence Q is not a subsequence of S^k. In order to find such a sequence, the query sequence Q should be $Q = \langle(\{1\},0),(\{5\},2)\rangle$ – the user must explicitly specify the number of sets after the first set in the query sequence.

Still, further research is required. Our future work agenda includes optimization of the physical structure of the index and designing efficient algorithms for index maintenance. Inserting and deleting of sequences from the index is not supported yet. Creating of new algorithms for insertion and deletion of sequences is our next goal.We also plan to run excessive experiments on real-world data sets to prove the practical usability of the proposed index, as well as creating an index that would be free of the aforementioned problem.

References

1. R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, pages 69–84. Springer-Verlag, 1993.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.
3. W. Andrzejewski, P. Gaertig, M. Radom, and M. aj Antoniewicz. *Opracowanie i analiza wydajnościowa indeksu dla przybliżonego wyszukiwania podzbiorów danych*, pages 44–54. 2003.

4. M. Araujo, G. Navarro, and N. Ziviani. Large text searching allowing errors. In R. Baeza-Yates, editor, *Proceedings of the 4th South American Workshop on String Processing*, pages 2–20, Valparaiso, Chile, 1997. Carleton University Press.
5. J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435, New York, NY, USA, 2002. ACM Press.
6. U. Deppisch. S-tree: a dynamic balanced signature index for office retrieval. In *SIGIR '86: Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 77–87, New York, NY, USA, 1986. ACM Press.
7. C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 419–429. ACM Press, 1994.
8. J. M. Hellerstein and A. Pfeffer. The rd-tree: an index structure for sets. Technical Report 1252, University of Wisconsin at Madison, 1994.
9. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
10. Y. Ishikawa, H. Kitagawa, and N. Ohbo. Evaluation of signature files as set access facilities in oodbs. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 247–256, New York, NY, USA, 1993. ACM Press.
11. E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 151–162. ACM Press, 2001.
12. E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM Press, 2004.
13. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademia Nauk SSSR*, 163(4):845–848, 1965.
14. M.-Y. Lin and S.-Y. Lee. Fast discovery of sequential patterns through memory indexing and database partitioning. *J. Inf. Sci. Eng.*, 21(1):109–128, 2005.
15. N. Mamoulis and M. L. Yiu. Non-contiguous sequence pattern queries. In *Proceedings of the 9th International Conference on Extending Database Technology*, 2004.
16. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 319–327. Society for Industrial and Applied Mathematics, 1990.
17. E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.
18. A. Nanopoulos, Y. Manolopoulos, M. Zakrzewicz, and T. Morzy. Indexing web access-logs for pattern queries. In *WIDM '02: Proceedings of the 4th international workshop on Web information and data management*, pages 63–68. ACM Press, 2002.
19. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224, Washington, DC, USA, 2001. IEEE Computer Society.

20. R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT '96: Proceedings of the 5th International Conference on Extending Database Technology*, pages 3–17, London, UK, 1996. Springer-Verlag.
21. E. Ukkonen. Constructing suffix trees on-line in linear time. In J.v.Leeuwen, editor, *Information Processing 92, Proc. IFIP 12th World Computer Congress*, volume 1, pages 484–492. Elsevier Sci. Publ., 1992.
22. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
23. M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series with support for multiple distance measures. In *ACM KDD*, 2003.
24. H. Wang, C.-S. Perng, W. Fan, S. Park, and P. S. Yu. Indexing weighted-sequences in large databases. In *Proceedings of International Conference on Data Engineering*, 2003.
25. P. Weiner. Linear pattern matching algorithms. In *Proceedings 14th IEEE Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.
26. B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 385–394. Morgan Kaufmann Publishers Inc., 2000.
27. M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.